Analysis of an automatic grading system within first year Computer Science programming modules

Emlyn Hegarty-Kelly
Department of Computer Science, Maynooth University
Maynooth, Ireland
emlyn.hegartykelly@mu.ie

Dr Aidan Mooney
Department of Computer Science, Maynooth University
Maynooth, Ireland
aidan.mooney@mu.ie

ABSTRACT

Reliable and pedagogically sound automated feedback and grading systems are highly coveted by educators. Automatic grading systems are useful for ensuring equity of grading of student submissions to assignments and providing timely feedback on the work. Many of these systems test submissions to assignments based on test cases and the outputs that they achieve, while others use unit tests to check the submissions.

The approach presented in this paper checks submissions based on test cases but also analyses what the students actually wrote in their code. Assignment questions are constructed based around the concepts that the student are currently learning in lectures, and the patterns searched for in their submissions are based on these concepts. In this paper we show how to implement this approach effectively. We analyse the use of an automatic grading system within first year Computer Science programming modules and show that the system is straightforward to use and suited for novice programmers, while providing automatic grading and feedback.

Evaluation received from students, demonstrators and lecturers show the system is extremely beneficial. The evaluation shows that such systems allow demonstrators more time to assist students during labs. Lecturers can also provide instant feedback to students while keeping track of their progress and identifying where the gaps in students' knowledge are.

KEYWORDS

Automated Assessment, Computer Science Education, CS1, Coding

ACM Reference Format:
Emlyn Hegarty-Kelly and Dr Aidan Mooney. 2021. Analysis of an automatic grading system within first year Computer Science programming modules. In *Computing Education Practice 2021 (CEP '21), January 7, 2021, Durham, United Kingdom.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3437914.3437973

1 INTRODUCTION

Automatic grading of student submissions is often seen as an ideal rather than a reality. Can an automated system perform equally as well, or better, than a human grader? Within Computer Science (CS) students tend to be exposed initially to CS1 which focuses on programming. CS has a notoriously high failure rate at the end of first year with programming modules seen as a major stumbling block. In Ireland, the non-progression rate for first year CS students is currently at 25%, the highest among all disciplines in higher education [6].

Additionally, CS1 tends to have high student numbers and programming is an individual task and can be very frustrating, with struggling students feeling isolated and often embarrassed to ask questions, and it is not until the final exam that they may be identified as struggling [10]. The CS1 class discussed within the paper was a first year university class with approximately 400 students and covered topics from variables up to multi-dimensional arrays in Java. The class consists of mainly school leavers and is comprised of students doing a wide cohort of degree programmes, across both Science and Arts fields.

Grading CS1 coding assignments can be an unwieldy process. Students may submit their assignments online or they are corrected in labs by demonstrators. These methods have their own inherent problems. When assignments are submitted online they need to be sorted and organised to ensure they are in the correct file type and format, they then need to be compiled and run on the graders computer before finally having a mark assigned. When they are corrected in labs by demonstrators it is difficult to ensure that grading is consistent and unbiased when there are multiple demonstrators. While this grading is taking place, students are missing out on vital help and support that the demonstrator could be providing. Both of these grading processes are difficult with small class sizes and become more onerous and time consuming as the class size increases. The way to overcome the inherent difficulties with these processes is to incorporate an automated grading system.

An automated grading system which is unbiased is extremely beneficial to students and educators. All students within a class group will be graded similarly ensuring that no bias occurs; in comparison to the case where multiple demonstrators might be reviewing work within a class. When the student submits the assignment, the files are required to be in the correct type and format, and the file will be automatically compiled and executed by the system with an associated grade being assigned. The time to complete the entire grading process is also greatly reduced. Wilcox [14] looked at "The role of automation in undergraduate computer science education" and concluded that automation can save significant time and resources without negatively impacting the academic performance of students in introductory courses. However, as highlighted by Wrenn et al. [15] automated grading systems do have their place, but we must reflect on their usage regularly to determine if they are actually benefiting students.

CEP '21, January 7, 2021, Durham, United Kingdom

ACM ISBN 978-1-4503-8959-4/21/01…$15.00
https://doi.org/10.1145/3437914.3437973

There are a number of approaches that one could take when looking at the automatic grading of student assignments and these will be discussed in more detail in Section 2. This paper looks at our approach to automated grading of student assignments in their introductory programming module during weekly lab assignments. Within our CS1 module, and subsequent CS2 module, students are encouraged to continue to work outside of scheduled class and lab times. They have access to an automated grading system to redo past assignments and practice their programming at all times, all of which aids in their learning. The work presented in this paper is early stage research and we are currently enhancing the system with the intention of bench-marking the effectiveness of the system in the near future.

2 RELATED WORK

Automatic grading has received a lot of attention in CS education [7] and as such many different systems have been developed and used to supplement teaching within CS. Each of these systems provide ways for students to store and maintain their code, but they each use different methods to grade the students work.

There are a number of approaches that can be taken with an automatic grading system. One approach taken looks at using certain inputs to a program and determining if the output of a students program matches the expected output, taking in to account the functionality that is expected from the assessed program. Examples of this are repl.it classroom [11], Stepik [13] and Autolab Project [1]. These systems allow students to write code in an online editor and submit it for grading. Automatic grading is then completed by having predetermined test cases in the form of input values and when the code runs it is matched against expected output values. A calculated grade is then assigned to this submission on this basis.

Another approach is to allow students to test their own code using tools like Web-CAT [4]. When a student is submitting code in Web-CAT they must also supply test cases via JUnit. The JUnit test cases, provided by the student, ensure the solution addresses all of the requirements of the question being asked and that the solution is valid. OK [9] is a system that combines both approaches mentioned above but provides more graphical feedback for graders about how many students submitted solutions and the grades achieved, among other statistics. DeNero, Sridhara, Pérez-Quiñones, Nayak, and Leong [3] carried out an overview of automatic grading systems, including Autolab Project, Web-CAT and OK, where they identify the structure of each of the systems and how they operate. HackerRank [5] is a professional automatic assessment platform that is used by technology recruiters and hiring managers to objectively evaluate developer skills. The platform can be used to assess developers skills based on their program outputs. A system like HackerRank has many advantages but it does not provide the same level of support for novice programmers as other tools.

All of the systems presented provide automatic grading solutions for different aspects of CS modules, from code output to unit tests and algorithm visualisations. These systems however, either just test against a suite of test cases for a solution, which does not take into account the students code, or look for unit tests that students must also write for their code. This can be problematic for novice CS students who might have difficulty in grasping the core concepts of writing a program and do not need these extra complexities.

Another approach to automatic grading uses pattern matching techniques to looks for certain constructs in the code. For example, an educator may want students to use a for loop and an array structure, and pattern matching techniques can be used to look for these in the code submission. A combination of the discussed techniques can be used to allow for as much coverage of student submissions as possible. Our solution uses this pattern matching technique, with novice CS students, to assist them in core programming concepts and give them feedback based on what they have written in their submitted code as well as testing for correctness with test cases, which check the output from the submissions.

3 BASH SCRIPTING

The automatic grading system operates using Bash scripting. This section describes in detail how the scripts have been continually developed and improved upon to incorporate new features and will present how the automatic grading is carried out.

Initially, the scripts started out in their simplest form and running the students code against a series of test cases. The system would then generate a grade based on the number of test cases that the code passed. However, no consideration was made in this approach for the work that the student had put into their solution when it did not pass the test cases. With this in mind further work was undertaken on the scripts to not only look at the output from student's code but also to look at what they wrote in their solutions. The steps undertaken were as follows:

(1) Compile the students code and check for errors,
(2) Use grep functionality to identify patterns that match to concepts being assessed,
(3) Provide input for the test cases along with the expected output for each checking how many test cases the are passed,
(4) Assign marks for each of the 3 steps above,
(5) Determine appropriate feedback for the students.

There are two different frameworks that our automatic grading system has used. These are the Virtual Programming Lab (VPL) [12] and MULE [2]. VPL is a plugin for the Moodle learning environment, and MULE is a browser-based Integrated Development Environment (IDE). The frameworks allow the students to view course assignments and write their code in an online platform. Additionally, they allow students to practice their coding skills when not in a lab environment, and still receive feedback on their work.

3.1 Compile the code

The first step is to check if the code provided by the student as a solution compiles correctly. If there are errors in the compilation process the errors are outputted to the screen for the students to fix them. At this stage it is also important to check that the submission contains code and is not an empty file.

3.2 Grep and Regular Expressions

The next step is to use grep, which is a process within the bash scripting language allowing the system to search the code for patterns. The regular expressions (regex) to find these patterns would correspond to different concepts within the Java programming language. In initial iterations of the system there were regexs' developed that would not always properly capture a fully syntactically

correct construct. For example, in a "for loop" if a student uses commas instead of the required semi colons within the for loop block, the compilation process would fail as the syntax is incorrect but the student might get full marks for this incorrect code structure. As a result many of the original regexs' needed to be better structured to allow students flexibility in how they write their code but still ensuring to capture the concept being assessed, while being syntactically correct.

3.3 Test Cases

After compiling the code, it remains important to ensure that the student is on the correct learning path with correctly functioning code and to use test cases to check that the code functions correctly. These test cases should effectively check the student's solutions to ensure complete test coverage. In the output, the students are shown how their code handled each test case, and if they failed a test case they would be shown the expected result. This helps to provide the student with reasoning and feedback as to where they have gone wrong. To allow for students to solve the question in an unexpected way the number of test cases that their code passes is compared to the actual number of test cases. If the students code passes all test cases then full marks are awarded.

3.4 Assign Marks

Lab questions are created weekly based on the concepts being covered in class. Marks are awarded within the questions in three categories. The first category sees marks awarded for the successful compilation of their code. The second category sees marks awarded for matching the different code constructs for the topic being assessed. The third category sees marks awarded for passing the test cases for the question being asked.

3.5 The Feedback That Is Provided

At different stages in the process, feedback can be provided. If the student submission does not compile, standard error messages are displayed. If certain code patterns are not found in the solution a message can be displayed highlighting this. The system can provide the input and expected output for the test cases and determine if the solution passes these test cases. The current feedback system has proven to be successful in engaging students [8], but it was felt that the feedback could be improved and further developed.

4 EVALUATION

While maintaining and developing this automatic grading system evaluation from the different user types was gathered. These users are the students completing assignments, demonstrators who assist students during lab times and lecturers who set assignments. In this section we provide a review of these different types of evaluation.

4.1 Student Evaluation

Students were asked for their evaluation on using an automated grading system to do their weekly lab assignments. Participation in this survey was voluntary and some of the comments included:

- Instant feedback is extremely helpful understanding code.
- I feel it is an excellent piece of software to practice my code.
- Easier to plan and organise my code.
- Challenging, efficient, helpful to the module.
- Useful in the sense that it tells you what your code is missing.
- Having a built-in set of test cases meant the evaluation was a big help.

This collection of comments represents the positive experience of the automated grading system experienced within the class. There was some negative feedback to the system, mainly related to the description of errors provided by the tool, as summarised by:

- It would help more if it gave a more detailed description of errors.

4.2 Demonstrator Evaluation

Demonstrators work directly with the students in their weekly labs and have experienced the automatic grading system first-hand. The demonstrators were asked about the advantages and disadvantages of using the automatic grading system. In total seven demonstrators provided evaluation to the survey through Microsoft Forms.

When asked about the advantages of using the system and feedback provided, the following comments were received:

- Less biased and error prone correcting work, an automatic grading can resolve some trivial issues reducing workload.
- With auto grading, more time is allowed for teaching rather than marking their answers.
- If the mistake was anticipated the feedback is good enough I don't need to help much, the students can make progress themselves. However, if the mistake was unexpected, it can be time consuming to find out where the problem lies.
- Failed test cases are very helpful to show the student where their code went wrong. Often the grading script will have better test cases than the student or I could have come up with it.

Overall 86% of demonstrators said the system was very helpful for the students in labs with the remaining 14% expressing no feelings either way. The demonstrators were also asked about disadvantages of the system and the common sentiment was:

- The system is not always capable of guiding the student to the correct answer. A demonstrator grading can usually give an excuse for an answer being incorrect. There are cases where the system provides a fail, but the reason is not obvious ..., sometimes leading to further confusion.

The demonstrators were asked "From a demonstrator point of view, does the feedback provided by the automatic grading system help you to help the students". Some of the responses were:

- It is quicker to see where they're going wrong.
- Yes, the system helps me find the type of error quickly and the test cases help me determine what inputs are causing the students program to fail.
- Yes, with the feedback, reading errors is encouraged and this shows them how to locate their mistake; showing them how to read their errors has been greatly helped by the feedback.

The demonstrators were asked "From a demonstrator's point of view, how useful do you find the automatic grading in labs? (1 - not useful at all and 5 - extremely useful). The average response here was 4.4 out of 5, highlighting the strong belief of demonstrator's that this tool is a significant benefit to them supporting students.

4.3 Lecturer Evaluation

Evaluation was received from three lecturers who regularly use automated grading tools, in order to gather evidence on their benefits. The automatic grading system was given a rating of 4.5/5 in terms of its usefulness to lecturers in their modules. The lecturers were also asked about the benefits of the system within their module for both demonstrators and students. The responses were:

- It has the potential to provided faster feedback (sometimes dynamic feedback) to the student. It guarantees that grading is consistent. The digital recording of the submission and grade data is useful for auditing and storing of assignments. Once developed it can save time which can be used for demonstrators and lecturers to use in the lab for discussion.
- Yes, the quick feedback and time saved allows for more discussion in the lab sessions.
- Yes definitely. The demonstrator can use the grading received by a student to direct their support and not get lost in trying to find errors within the code.
- The system removes the need for the command line interface required by MASM (and C), this allows students to focus on concepts and for demonstrators to do the same. The quicker code/compile/review cycle also maintains focus on the assignment in the lab.

The main disadvantage that the lecturers discussed was around the set up time of the scripts for the assignments but they commented that once a script was created it was relatively easy to transfer it across to other questions with minor adjustments. The lecturers were also asked about what improvements could be made to the system. The responses to this question were as follows:

- An easier method of creating scripts.
- The system should look at the totality of the submission, code submitted, compile and run time and test cases. With code it should be aware of industry standards (e.g. indentation, etc) and algorithms (e.g. detect quick sort or bubble sort).
- Allow for a wider range of text/string answers (Which are obviously checked by more complex regular expressions).

The lecturers were asked if the feedback provided by the automatic grading system helped them in helping their students:

- Yes, the quick feedback and time saved allows for more discussion in the lab sessions.
- Indirectly as it is allowing the demonstrators to provide more time to them without spending time correcting work.

In additional evaluation, one lecturer referred to the increased support and scaffolding available for students as they have more demonstrator one-on-one time with the students in labs.

4.4 Discussion

Evaluation on our automated grading systems was very positive from all involved. Students appreciated the quick grading which allowed them to improve their code. It appears to motivate the students to strive towards passing all of the test cases by highlighting failed ones. Demonstrators are freed up to spend more time helping the students and the automated grading tool allows them to quickly see where they should target their support. Lecturers noted that the consistency in grading is a key feature for these tools as all students are grading equally. In addition, the recording of all submissions and grades aids easy auditing of modules.

5 CONCLUSIONS AND FUTURE WORK

Automatic grading of student work can free up other resources in class like demonstrator time. This allows demonstrators to spend time helping students with their coding issues rather than spending time grading their work. Automated grading tools also ensure that all students are graded using the same rubric, thus ensuring equality and non-bias while also ensuring a high standard of work.

We have been using automated tools for a number of years. The positive feedback voiced by students, demonstrators and lecturers alike have convinced us to invest more time and effort enhancing these tools. We endeavour to enhance these tools by refining and improving the already created scripts, in addition to adding more questions to our bank of scripts. With our system the valuable and timely feedback for students is most important, and with this in mind we are beginning a large scale study of over 400 CS1 students. We will analyse each student's code every time they save and evaluate work while answering questions, to gain strong insights into how the students react to feedback and errors in real time. We hope this investigation will help us further refine our system and to help students get the most from this novel approach to automatic grading for CS1 and CS2 class groups. The main potential effects of this intervention would be to see a higher student engagement with the course material through tailored feedback in the assessments and increased retention within the modules.

REFERENCES

[1] Autolab Project. 2020. Autolab Project. http://www.autolabproject.com Accessed: 2020-06-11.

[2] Natalie Culligan and Kevin Casey. 2018. Building an Authentic Novice Programming Lab Environment. In *International Conference on Enguaging Pedagogy (ICEP)*.

[3] John DeNero, Sumukh Sridhara, Manuel A Pérez-Quiñones, Aatish Nayak, and Ben Leong. 2017. Beyond Autograding: Advances in Student Feedback Platforms.. In *SIGCSE*. 651–652.

[4] Stephen H Edwards and Manuel A Perez-Quinones. 2008. Web-CAT: automatically grading programming assignments. In *ACM SIGCSE Bulletin*, Vol. 40. ACM, 328–328.

[5] HackerRank. 2020+. HackerRank. https://www.hackerrank.com Accessed: 2020-07-14.

[6] David Harmon and Stephen Erskine. 2017. Eurostudent Survey VI. http://hea.ie/assets/uploads/2018/01/HEA-Eurostudent-Survey.pdf

[7] Jack Hollingsworth. 1960. Automatic graders for programming classes. *Commun. ACM* 3, 10 (1960), 528–529.

[8] Aidan Mooney, Susan Bergin, and Emlyn Hegarty-Kelly. 2017. Incorporating the Virtual Programming Lab into a first year Computer Science module. In *Technology-Enabled Feedback Approaches for First-Year: Y1Feedback Case Studies in Practice.*

[9] OK. 2020. OK. https://okpy.org Accessed: 2020-07-11.

[10] Keith Quille, Susan Bergin, and Aidan Mooney. 2015. Press#, a web-based educational system to predict programming performance. *International Journal of Computer Science and Software Engineering (IJCSSE)* 4, 7 (2015), 178–189.

[11] Replit. 2020. repl.it classroom. https://repl.it/site/classrooms Accessed: 2019-06-11.

[12] Juan Carlos Rodríguez-del Pino, Enrique Rubio Royo, and Zenón Hernández Figueroa. 2012. A Virtual Programming Lab for Moodle with automatic assessment and anti-plagiarism features. (2012).

[13] Stepik. 2020. Stepik - smart tools for IT instructors. https://stepik.org/catalog Accessed: 2020-07-11.

[14] Chris Wilcox. 2015. The role of automation in undergraduate computer science education. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*. ACM, 90–95.

[15] John Wrenn, Shriram Krishnamurthi, and Kathi Fisler. 2018. Who Tests the Testers? *(ICER '18)*. Association for Computing Machinery, New York, NY, USA, 51–59.